EVOCHAMBER: Test-Time Co-evolution of Multi-Agent System at Individual, Team, and Population Scales

Yaolun Zhang[1,5,*], **Tianyi Xu**[2,*], **Shengyu Dai**[3]
Zhenwen Shao[3], **Qingyun Wu**[4,5], **Huazheng Wang**[1,5]

[1]Oregon State University [2]University of Wisconsin–Madison
[3]Johnson & Johnson [4]Pennsylvania State University [5]AG2AI, Inc.

{zhanyaol, huazheng.wang}@oregonstate.edu, txu223@wisc.edu
{SDai9, ZShao5}@its.jnj.com, qingyun.wu@psu.edu

[*]Equal contribution.

Abstract

We argue that multi-agent test-time evolution is not single-agent evolution replicated N times. A single-agent learner can only evolve its own context and memory. A multi-agent system additionally evolves who collaborates, how they collaborate, and how knowledge flows across the population. These components have no single-agent counterpart and can produce phenomena such as emergent specialization. Yet prior test-time methods either confine experiences to individual agents, forfeiting cross-agent learning, or broadcast symmetrically to all agents, erasing the specialization that makes collaboration valuable. We present EVOCHAMBER, a training-free framework that instantiates test-time evolution at three levels over a coevolving agent pool. At its core is CODREAM (**Co**llaborative **Dream**ing), a post-task protocol triggered on team failure or disagreement, in which agents collaboratively reflect, distill insights, and route them asymmetrically from strong to weak agents on the failed niche, preserving specialization while filling knowledge gaps. Team-level operators assemble niche-conditioned teams and select collaboration structures online. Population-level lifecycle operators fork, merge, prune, and seed agents under performance pressure. On three heterogeneous task streams with Qwen3-8B, EVOCHAMBER reaches 63.9% on competition math, 75.7% on code, and 87.1% on multi-domain reasoning, outperforming the best baseline by 32% relative on math and confirming asymmetric cross-agent transfer as the primary driver in ablation. Starting from several identically initialized agents, four to five stable niche specialists spontaneously emerge, a structural signature of multi-agent evolution that no single-agent learner can express. See our code at: https://github.com/Mercury7353/EvoChamber

1 Introduction

Large Language Models (LLMs) [21] excel at reasoning [35], coding, and recall. Multi-agent systems (MAS) built on LLMs assign roles and communication patterns across multiple LLM instances [11, 25, 15, 19, 36]. Deployed over continual task streams, such systems should improve with experience: breakthroughs should inform later tasks, and recurring task types should be routed to the best-suited agents.

However, evolving a multi-agent system is fundamentally different from evolving a single agent N times in parallel. A single-agent learner, such as Reflexion [28] or ExpeL [43], evolves only one

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

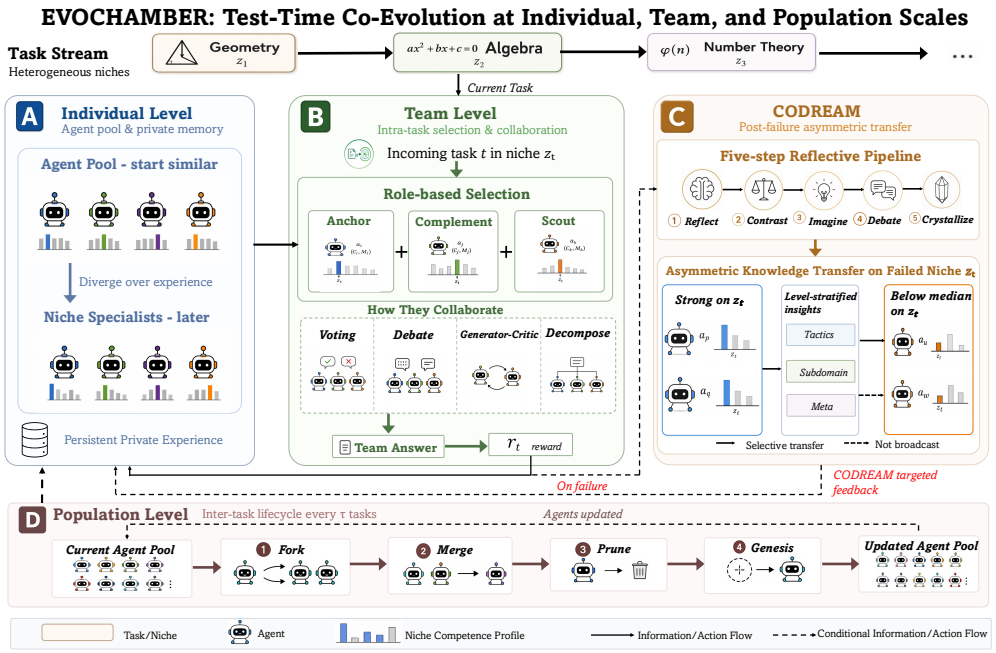


Figure 1: Overview of EvoChamber. Starting from a pool of N identically initialized agents (*individual level*), a niche-conditioned selector assigns three functional roles, anchor, complement, and scout, and a leader-learned policy selects one of four collaboration structures. The team outcome is attributed as a shared reward (*team level, intra-task*). Between tasks, asymmetric transfer (CoDream) routes insights from high-fitness to deficit agents, and lifecycle operators fork, merge, prune, and seed new agents to edit pool composition (*population level, inter-task*).

agent's context and memory. A multi-agent system, in contrast, maintains a pool of agents and a strictly richer evolvable state. Beyond the individual level, the state includes a *team* component that determines who collaborates, how they collaborate, and how the joint outcome updates per-agent knowledge. It also includes a *population* component that governs knowledge flow between agents and edits pool membership over time, producing phenomena such as emergent specialization that have no counterpart for a single agent.

Yet existing work does not instantiate this full state space. Methods that evolve individual agents, including EvoMem [9] and MemCollab [2], confine experiences to one agent or broadcast them symmetrically to all agents. The former forfeits cross-agent learning and the latter erases specialization, because every agent receives identical memory regardless of individual strengths. A parallel line of work pursues multi-agent co-improvement through RL fine-tuning [37, 24, 5] or offline structure search [13, 42, 41], but these methods operate on fixed agent roles within a single domain and freeze the resulting system at deployment. Neither camp addresses the question: *how can a multi-agent system continuously evolve at test time, across heterogeneous task streams, without gradient updates?*

To investigate this question, we propose **EvoChamber**, a training-free framework that instantiates test-time evolution on all three levels over a coevolving agent pool (Fig. 1). At the individual level, every agent accumulates private experience and niche competence estimates. At the team level, a niche-conditioned selector assembles a team of three complementary agents and a leader selects one of four collaboration structures online. At the population level, CoDream (**Co**llaborative **Dream**ing) triggers on team failure or disagreement: agents collaboratively reflect, distill insights, and route them asymmetrically from strong to weak agents on the failed niche, preserving specialization while filling knowledge gaps. Lifecycle operators periodically fork, merge, prune, and seed agents under performance pressure. Table 1 positions EvoChamber against prior work along the three evolution levels.

	Individual		Team (intra-task)		Population (inter-task)	Online	Training
Method	context	memory	composition	structure	transfer / pool edit		free
Reflexion [28]	✓	✓	×	×	× / ×	✓	✓
MemCollab [2]	✓	✓	×	×	sym. / ×	✓	✓
CoMAS [37]	✓‡	×	×	×	× / ×	×	×
MAPoRL [24]	✓‡	×	×	✓	× / ×	×	×
EvoMAC [13]	✓	×	✓	✓	× / ×	✓§	✓
AFlow [42]	×	×	✓†	✓†	× / ×	×	✓
EvoChamber (ours)	✓	✓	✓	✓	✓ / ✓	✓	✓

‡ CoMAS and MAPoRL update weights via RL rather than evolving at test time. § EvoMAC adapts within one task only. † AFlow's structure search is offline and frozen at inference.

Table 1: Evolution levels activated by representative methods. EvoChamber is the first to activate all three levels online without training. See §2 for extended discussion.

We evaluate EvoChamber on three heterogeneous task streams and two model families. With Qwen3-8B, EvoChamber reaches 63.9% on Hard Math, 75.7% on Hard Code, and 87.1% on AFlow-Stream, outperforming the best baseline MemCollab by 32% relative on math and achieving a $5\times$ improvement on CodeContests over a single agent. Gains are largest in the hardest regimes and transfer to GPT-4.1-mini. Ablations that disable the team or population level yield level-specific drops, with the single largest drop of -10.8% from removing CoDream, confirming asymmetric cross-agent transfer as the primary driver. Beyond aggregate accuracy, we observe a signature that is structurally impossible for any single-agent learner: starting from several identically initialized agents, four to five stable niche specialists spontaneously emerge, and this pattern is reproducible across random seeds even though the identity of each specialist changes.

2 Related Work

Static multi-agent systems. AutoGen [36], MetaGPT [11], CAMEL [15], DyLAN [19], AgentVerse [4], and Mixture-of-Agents [31] assign fixed or dynamically grouped roles, but agent knowledge cannot evolve with the task stream. Multi-agent debate [7, 17] and test-time reasoning enhancements [40, 29] improve answer quality but carry no persistent state across tasks. AFlow [42], Archon [27], ADAS [12], and ScoreFlow [34] discover workflows or agent architectures offline via search, while GPTSwarm [44] and MacNet [26] optimize multi-agent graphs via gradient signals, yet the result is frozen at inference time. EvoMAC [13] adapts agent interactions within a single task but does not carry experience across tasks. EvoChamber is complementary: where automated design optimizes *workflow graphs* offline, EvoChamber evolves *agent content* online.

Individual agent memory. Self-Refine [20] iterates on a single agent's output through self-feedback, Reflexion [28] accumulates self-critiques, ExpeL [43] extracts reusable insights from trajectories, and AgentNet [38] equips agents with personal RAG stores. EvoMem [9] extends Reflexion-style memory to a pool setting. All improve individual agents but provide no mechanism for one agent's learning to transfer to another, which is critical at low success rates where individual memory accumulates mostly failures.

Symmetric shared memory. MemCollab [2] distills team trajectories into a shared store broadcast to all agents, enabling collective learning, but the sharing is symmetric: every agent receives identical memory regardless of individual strengths, conflating domain-specific strategies and destroying specialization. EvoChamber's CoDream addresses this through asymmetric, gap-targeted distillation that routes insights only to deficit agents.

Gradient-based co-evolution. CoMAS [37] co-evolves agents via interaction rewards, MAPoRL [24] applies multi-agent post-co-training with RL, MAE [5] pursues LLM self-improvement through co-evolution, and MAS^2 [32] specializes agents via DPO. These methods require gradient updates on a static training distribution. EvoChamber achieves comparable qualitative goals through inference-time prompt evolution alone. No prior work simultaneously achieves pool-level persistent

state, verified asymmetric cross-agent distillation, and structural pool evolution, all without gradient updates and all online (Appendix C).

3 Method

3.1 Problem Formulation and the Solve-Evolve Loop

Let $\mathcal{T} = (t_1, \ldots, t_T)$ be an online stream of tasks drawn from K heterogeneous niches, with per-task niche label z_t and reward $r_t \in [0, 1]$. The objective is to maximize $\sum_t r_t$ by evolving system state.

The per-task loop. For each task t, EVOCHAMBER (i) *selects a team* of three agents with roles anchor, complement, scout. (ii) The anchor (also leader) chooses structure L_t from its experiences. (iii) The team *executes* L_t, scoring as r_t. (iv) r_t *propagates* as a shared reward, updating per-agent competence and pool-wide pair synergy. On failure or disagreement, a post-hoc CODREAM session emits insights to deficit agents. (v) Every τ tasks, lifecycle operators (fork, merge, prune, genesis) edit pool membership.

3.2 What Evolves: Three-Level State Decomposition

A single-agent learner evolves only $\theta_t^{\text{SA}} = (C_t, M_t)$, where C_t is the working context and M_t is the persistent store retrieved into C_t. A multi-agent system maintains a pool $\mathcal{P}_t = \{a_1, \ldots, a_{|\mathcal{P}_t|}\}$ and a richer evolvable state

$$\theta_t^{\text{MAS}} = \underbrace{\{(C_t^i, M_t^i)\}_{i \in \mathcal{P}_t}}_{\text{individual}} \oplus \underbrace{(T_t, L_t)}_{\text{team (intra-task)}} \oplus \underbrace{(\Sigma_t, \Omega_t, \mathcal{P}_t)}_{\text{population (inter-task)}}, \tag{1}$$

where T_t is the size-k team selected for task t and L_t is the collaboration structure used to combine its outputs. The remaining three quantities persist across tasks and drive how teams are formed.

Pair-wise synergy Σ_t captures whether agents i and j work well together on niche z, a question no per-agent statistic can answer. We maintain $\Sigma_t[i, j, z] = \sigma_{ij}(z)$ as the running mean team reward over past niche-z tasks in which i and j co-participated. Composition (§3.4) reads Σ_t to favor complements with high prior synergy with the anchor.

Pair-wise style overlap Ω_t prevents teams of strong but redundant agents. We define $\Omega_t[i, j] = \omega_{ij} = \cos(\vec{q}_i, \vec{q}_j)$, the cosine similarity between niche-competence vectors $\vec{q}_i = (q_i(z_1), \ldots, q_i(z_K))$. Composition penalizes high ω when adding members, biasing teams toward complementary skill profiles. Ω_t is derived from $\{\vec{q}_i\}$ and requires no separate update.

Mutable roster $\mathcal{P}_t$ is the set of active agents, with $|\mathcal{P}_t| \gg k$ so that selection has room to maneuver. $\mathcal{P}_t$ is itself evolvable: lifecycle operators (§3.5) periodically fork, merge, prune, and seed agents, so the pool's *shape*, not just its members' memories, adapts to the task stream.

Figure 2 illustrates the gap on a single task: a single (C, M) produces one trajectory and one answer, while the multi-agent state routes the same task to three agents with different accumulated histories, aggregates their perspectives through a task-chosen structure, and updates $(\Sigma, \Omega, \mathcal{P})$ as a side effect. The next three subsections detail each level.

3.3 Individual-Level Evolution

The individual level maintains each agent's private knowledge: its accumulated experience and niche competence.

Experience archive. After each task in which a_i participates, a_i reflects on its intermediate outputs, the team's answer, and the reward. The reflection produces two lessons at different granularities: a *subtask-level* lesson indexed by the niche label z_t, and a *cross-domain meta-insight* not tied to any niche. Subtask lessons are bucketed by niche, meta-insights form one pool, and both grow with the agent's full history without capacity limit. At solve time, a_i retrieves the top-k entries from its niche-z_t bucket and meta-insight pool by cosine similarity over task embeddings, and prepends them to the prompt. This reflection is independent of LeadLearn (§3.4): one tracks how to solve, the other how to organize collaboration.

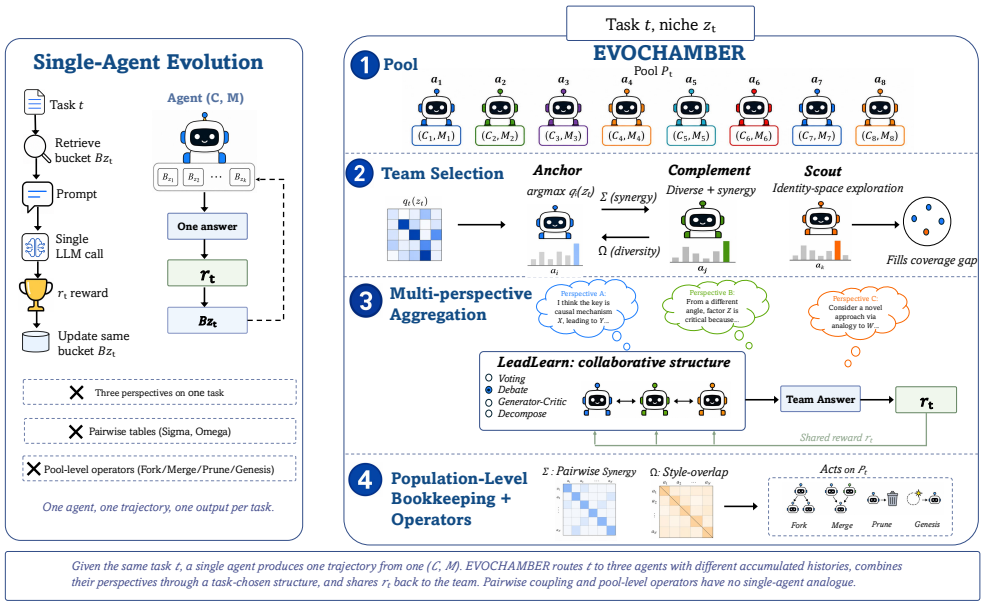


Figure 2: Same task t, two treatments. **Left:** a single agent produces one trajectory from one memory store. **Right:** three agents drawn from a pool of N heterogeneous histories, aggregated by a leader-chosen structure. Shared reward updates (Σ, Ω), and lifecycle operators edit $\mathcal{P}_t$ every τ tasks.

Niche competence. Beyond textual experience, each agent also tracks a scalar competence $q_i(z) \in [0, 1]$ estimating its expected reward on niche-z tasks. After each task with outcome $r_t \in [0, 1]$, we update via EWMA:

$$q_i(z) \leftarrow (1 - \alpha)\, q_i(z) + \alpha\, r_t, \tag{2}$$

initialized at $q_i(z) = 0.5$. EWMA is preferred over a running mean because competence is non-stationary as the agent's experience and teammates evolve, so recent outcomes carry more signal.

3.4 Team-Level Evolution

The team level assembles an agent team for each incoming task and decides how they collaborate. Individual heterogeneity emerges here: agents diverge only because team selection routes them to different task histories.

Composition: anchor, complement, scout. Picking the top three agents by $q_i(z_t)$ collapses diversity: strong agents accumulate all experience, weak agents never participate, and the pool loses the variety that lifecycle operators rely on. We therefore decompose the team into three roles with distinct selection rules. The **anchor** is the niche's current best performer,

$$a_t = \arg\max_{i \in \mathcal{P}} q_i(z_t), \tag{3}$$

with ties broken uniformly at random. It also serves as leader, avoiding a separate election. The **complement** is then drawn from the remaining pool $\mathcal{P} \setminus \{a_t\}$ to supply capability the anchor lacks:

$$c_t = \arg\max_{i \in \mathcal{P} \setminus \{a_t\}} \lambda_q\, q_i(z_t) + \lambda_\sigma\, \sigma_{i,a_t}(z_t) + \lambda_\omega\, (1 - \omega_{i,a_t}), \tag{4}$$

which jointly rewards own competence on z_t, prior synergy with the anchor on z_t, and stylistic distinctness from the anchor. The **scout** is drawn from the rest to enforce exploration and diversity:

$$s_t = \arg\max_{k \in \mathcal{P} \setminus \{a_t, c_t\}} \lambda_u\, u_k(z_t) + \lambda_d \left(1 - \bar{\omega}_{k,\{a_t,c_t\}}\right), \tag{5}$$

where $u_k(z_t) = 1/(1 + n_k(z_t))$ favors agents under-exposed on niche z_t and $\bar{\omega}_{k,\{a_t,c_t\}}$ is the mean style overlap with the two already-selected agents. This prevents collapse onto a few dominant

members by ensuring every agent periodically receives task experience. All weight coefficients $\lambda_{(\cdot)}$ are fixed across experiments.

Structure: LeadLearn. Once the team is fixed, the leader chooses a collaboration structure L_t from {**voting**, **debate**, **generator-critic**, **decompose**}. No single structure dominates across niches, so the leader learns this choice online. The pool maintains a *shared* experience bank of past leadership rounds, each entry a tuple *(team profile, task profile, structure, outcome, reflection)*. Sharing the bank lets (team, task)→structure meta-knowledge accumulate as the anchor rotates. At decision time, the leader forms a query vector ξ_t from the niche label and team competence profile, retrieves top-k entries by cosine similarity, and conditions the backbone LLM on these to propose L_t. After the task, the leader appends a new tuple with a short natural-language note on why L_t succeeded or failed, giving the bank a richer signal than scalar rewards alone.

Updates. After each task, all three agents update $q_i(z_t)$ via EWMA and increment $n_i(z_t)$. Pair synergy is updated analogously,

$$\sigma_{ij}(z_t) \leftarrow (1-\beta)\,\sigma_{ij}(z_t) + \beta\, r_t, \tag{6}$$

since pair compatibility is non-stationary as the agents evolve. The style overlap ω_{ij} is recomputed from the updated skill profiles. The leader's LeadLearn update is described above.

3.5 Population-Level Evolution

Two gaps remain after the individual and team levels: a useful lesson discovered by a strong agent stays inside that agent, and the pool's roster is itself a state that should evolve as new task types appear or old strengths become redundant. CODREAM addresses the first by routing knowledge between existing agents, while the lifecycle edits pool membership.

CODREAM: knowledge flow without dilution. A session fires whenever the team fails, either because the mean reward falls below threshold θ or because members disagree. The three team members run a five-phase reasoning loop: *Reflect* lets each member privately diagnose what went right or wrong in its own attempt. *Contrast* pairs failing members with successful ones to extract a delta, what the successful approach did differently. *Imagine* turns those deltas into hypothetical strategies tagged with the niches they might apply to. *Debate* has the members cross-critique each other's proposals, dropping weak ones. *Crystallize* converts surviving proposals into structured insights, each tagged with a level (task-local, subdomain-scoped, or cross-domain) and a niche scope. The insight is then written into every agent whose competence on that niche falls below the pool median. Strong agents thus produce knowledge while weak ones consume it, sharpening specialization rather than diluting it, the failure mode of symmetric broadcast [2].

Lifecycle: the pool roster as a variable. Every τ tasks the system inspects the pool and applies four operators, each targeting a different pathology of a static roster. **Genesis** fills coverage gaps: when a recurring task type has no specialist, a fresh agent is spawned from the most generalist parent with a persona aimed at the new type. **Fork** provides specialist headroom: when an agent dominates one task type, the system clones it with a persona mutation that further emphasizes that subdomain, preserving the parent. **Merge** removes duplication: when two agents have nearly identical skill profiles, they are consolidated, freeing a slot. **Prune** removes dead weight: an agent whose recent score lags the pool mean over a sustained window is retired. A fifth operator, *specialize*, nudges a high-performing agent's persona toward its dominant niche without changing the roster, so future selections sharpen the same agent rather than scattering experience.

The two halves of population-level evolution are decoupled: CODREAM continuously moves *what is known* between agents, while the lifecycle periodically reshapes *which agents exist*. Because $|\mathcal{P}| > k$, unused agents retain their state, so the pool carries old specialists alongside newly seeded ones without overwriting either.

4 Experiments

We evaluate EVOCHAMBER on three heterogeneous task streams and two model families, then verify robustness, decompose contributions via ablations, and analyze how the pool evolves.

Table 2: Hard Math Stream accuracy on Qwen3-8B. math_hard: 262 MATH Level 4/5; AIME'22–'25: 30 problems each; Overall: micro-average over 382 tasks.

Method	math_hard	AIME'22	AIME'23	AIME'24	AIME'25	Overall
SA	0.374	0.133	0.100	0.133	0.167	0.298
SC (k=5)	0.542	0.033	0.133	0.233	0.067	0.390
DyLAN	0.542	0.033	0.067	0.167	0.133	0.403
AgentNet	0.496	0.267	0.167	0.200	0.267	0.414
EvoMem	0.553	0.133	0.133	0.267	**0.300**	0.445
MemCollab	0.603	0.233	0.167	0.267	0.233	0.484
EvoChamber	**0.763**	**0.400**	**0.333**	**0.433**	**0.300**	**0.639**

4.1 Setup

Datasets. We construct three task streams that span different difficulty regimes and domain compositions. The Hard Math Stream combines 262 MATH [10] Level 4/5 problems with 30 problems from each of AIME 2022–2025, totaling 382 tasks. The Hard Code Stream contains 257 MBPP+ [1, 18], 164 HumanEval [3], and 165 CodeContests [16] problems, totaling 586 tasks that test whether debugging experience transfers across problem classes. The AFlow-Stream presents six domains in sequential 100-task blocks: GSM8K [6] $\rightarrow$ HotpotQA [39] $\rightarrow$ MBPP $\rightarrow$ MATH $\rightarrow$ HumanEval [3] $\rightarrow$ DROP [8], totaling 600 tasks that test adaptation under cross-block domain shifts. Each task carries a niche label z_t derived from its dataset metadata: MATH Level 4/5 vs. each AIME year for Hard Math, source benchmark for Hard Code, and domain block for AFlow-Stream. These labels index the per-niche competence statistics in §3.

Baselines. We compare against methods spanning different evolution levels. As no-evolution references, we include a stateless single agent (SA) and majority voting (SC, k=5) [33] as a compute-matched comparison. EvoMem [9] and AgentNet [38] evolve per-agent memory without cross-agent transfer, while MemCollab [2] extends this with symmetric pairwise sharing. DyLAN [19] adapts collaboration structures at inference time but maintains no cross-task state. All multi-agent baselines use k=3 agents to match our team size.

Implementation. EvoChamber uses N=20 identically initialized agents with team size k=3. The primary backbone is Qwen3-8B [30] served by 1 H100 GPU, and GPT-4.1-mini [22] from API for cross-backbone validation. A single hyperparameter configuration is used across all three streams and both model families with no per-benchmark tuning. See Appendix E.3.

Metrics. We report accuracy per stream: exact match for math, pass@1 for code, and F1 for QA.

4.2 Main Results

Tables 2–3 tell a consistent story across three streams: EvoChamber improves most where single-agent methods struggle, the advantage grows with task difficulty, and cross-agent knowledge transfer is what closes the gap.

Largest gains on the hardest tasks. On the Hard Math Stream (Table 2), EvoChamber reaches 0.639 overall, outperforming MemCollab by 32% relative and doubling the single-agent baseline. The gain concentrates where it matters most: +0.160 on math_hard and +0.167 on AIME'24. SC collapses on AIME to 0.067 because majority voting overrides rare correct outputs when per-agent accuracy is below 50%. EvoChamber avoids this by routing through a niche-competent anchor under a leader-selected structure.

Experience transfers across difficulty levels. On the Hard Code Stream (Table 3), MBPP+ saturates near 0.85 for all multi-agent methods. The discriminative subset is CodeContests, where EvoChamber reaches 0.352, a 5$\times$ improvement over a single agent. Debugging patterns learned on easier MBPP+ problems accumulate in agent profiles and propagate to deficit agents via CoDream, carrying over to the harder CodeContests problems. EvoMem and MemCollab score below SA on CodeContests at 0.027 and 0.084 respectively, suggesting that individual-level or symmetric memory alone introduces noise that hurts on the hardest problems without the niche-conditioned routing that CoDream provides.

Table 3: Accuracy on Hard Code Stream and AFlow-Stream. The HumanEval column is omitted from Hard Code as all methods score 1.000; Overall is the micro-average over all 586 tasks including HumanEval. The full breakdown is in Appendix G.

(a) Hard Code Stream

Method	MBPP+	CC	Overall
SA	0.842	0.068	0.667
SC (k=5)	0.849	0.198	0.708
DyLAN	0.825	0.189	0.695
AgentNet	**0.887**	0.102	0.698
EvoMem	0.885	0.027	0.672
MemCollab	0.870	0.084	0.682
EvoChamber	0.861	**0.352**	**0.757**

(b) AFlow-Stream

Method	GSM8K	HotpotQA	MBPP	MATH	HE	DROP	Overall
SA	0.960	0.791	0.780	0.780	0.800	0.800	0.819
SC (k=5)	0.890	0.778	0.560	0.610	0.410	0.690	0.656
DyLAN	0.670	0.888	0.690	0.620	0.830	0.840	0.756
AgentNet	0.970	0.820	0.793	0.680	**0.900**	0.800	0.827
EvoMem	0.940	0.892	0.817	0.660	0.880	0.850	0.840
MemCollab	0.960	0.847	0.793	0.660	0.890	0.840	0.832
EvoChamber	**0.980**	**0.895**	**0.843**	**0.820**	0.830	**0.860**	**0.871**

Table 4: Cross-backbone validation on the Hard Math Stream (top) and AFlow-Stream (bottom) under GPT-4.1-mini. The same hyperparameter configuration is used across both backbones and both streams.

Backbone (Stream)	Method	Subset Accuracy					Overall	Δ vs SA
		math_hard	AIME'22	AIME'23	AIME'24	AIME'25		
GPT-4.1-mini (Hard Math)	SA	0.824	0.400	0.300	0.333	0.367	0.675	—
	MemCollab	0.878	0.533	0.433	0.567	0.533	0.764	+0.089
	EvoMem	0.882	0.600	0.367	0.500	0.467	0.757	+0.082
	EvoChamber	**0.889**	**0.600**	**0.567**	0.533	**0.567**	**0.796**	**+0.121**

Backbone (Stream)	Method	GSM8K	HotpotQA	MBPP	MATH	HE	DROP	Overall	Δ vs SA
GPT-4.1-mini (AFlow-Stream)	SA	0.940	0.847	0.887	0.800	0.940	0.800	0.869	—
	MemCollab	0.950	0.864	0.910	0.680	0.940	0.850	0.866	−0.003
	EvoMem	0.940	0.896	0.910	0.680	0.950	0.860	0.873	+0.004
	EvoChamber	**0.950**	0.878	**0.960**	**0.820**	0.940	0.780	**0.888**	**+0.019**

Cross-domain adaptation across sequential domain blocks. On AFlow-Stream (Table 3), where six domains arrive in sequential 100-task blocks, EvoChamber reaches 0.871, ahead of EvoMem at 0.840 and MemCollab at 0.832. EvoChamber wins or ties on five of six domains, with the largest gains on MATH and MBPP where cross-agent coordination matters most. This stream tests exactly the scenario our three-level evolution is designed for: agents must specialize within domains while transferring metacognitive strategies across them.

Gains transfer across backbones and streams. Table 4 shows that the same hyperparameter configuration lifts EvoChamber above all baselines on both backbones and both streams. The absolute lift is larger when the backbone is weaker or the regime is harder: +0.341 on Qwen3-8B Hard Math, +0.121 on GPT-4.1-mini Hard Math, and +0.019 on GPT-4.1-mini AFlow, because GPT-4.1-mini's SA baseline on AFlow already reaches 0.869, leaving little headroom. EvoChamber remains the best method on both GPT-4.1-mini streams.

4.3 Ablation Studies

Table 5 decomposes contributions by evolution level on AFlow-Stream. The single largest drop comes from removing CoDream entirely: −0.108, establishing asymmetric cross-agent transfer

Table 5: (a) Ablation on AFlow-Stream: each row disables one of the method-level innovations EvoChamber introduces, mapped to the Method subsection that describes it. (b) Hard Math Stream under two random task permutations.

(a) Ablation on AFlow-Stream

Innovation (§)	Configuration	Acc.	Δ
—	EvoChamber (full)	0.871	—
Team composition (§3.4)	Random team (no niche-conditioned selector)	0.847	−0.024
Team structure (§3.4)	LeadLearn disabled (forced voting)	0.841	−0.030
Cross-agent transfer (§3.5)	− CoDream entirely	0.763	−0.108

(b) Hard Math Stream (permutations)

Condition	SA	EvoChamber	Δ
Default (fixed order)	0.298	0.639	+0.341
Shuffle (seed 42)	0.298	0.655	+0.357
Shuffle (seed 123)	0.298	0.662	+0.364

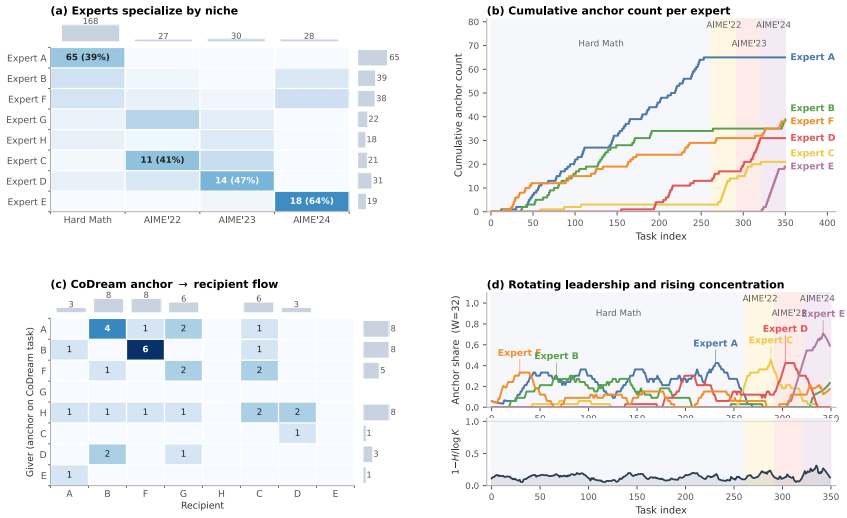


Figure 3: Four signals of pool co-evolution on the Hard Math Stream with Qwen3-8B, 382 tasks, seed 42. **(a)** Expert × niche anchor counts, column-normalized. **(b)** Cumulative anchor count per expert across the stream. **(c)** CoDream anchor → recipient flow matrix. **(d)** Rolling-window $W{=}32$ anchor share per expert (top) and pool-level specialization index $1{-}H(A^{\text{anchor}}\,|\,t)/\log N$ (bottom). For readability we relabel the top eight agents as Expert A through Expert H.

as the primary driver of collective learning. The effect is sharpest on dependent-reasoning domains where cross-agent coordination is essential, with HotpotQA dropping from 0.895 to 0.572 and DROP from 0.860 to 0.480. At the team level, disabling the niche-conditioned selector and disabling LeadLearn each produce independent drops of -0.024 and -0.030 respectively, confirming that team composition and team structure contribute separately. All innovations are non-redundant, and the gains decompose cleanly across the three evolution levels.

We also analyze the robustness of EvoChamber. Table 5 shows that under two independent random permutations of the Hard Math Stream, EvoChamber not only maintains its advantage over SA but actually improves slightly, reaching 0.655 and 0.662 compared to 0.639 under the default order. This rules out a favorable curriculum as the explanation: the gains come from the evolution mechanism, not task ordering. We further show in Appendix B that varying the initial pool size from $N{=}3$ to $N{=}20$ changes overall accuracy by only 0.011, as lifecycle operators grow or prune the pool to a similar effective size regardless of initialization.

4.4 Analysis: How the Pool Evolves

Figure 3 reports four signals extracted directly from the run log. Together they show that the pool co-evolves rather than converging to a static assignment, producing phenomena that no single-agent learner can exhibit.

Different niches acquire different specialists. Each niche column converges on a single dominant expert, and the dominant expert differs across AIME years. This niche separation cannot come from the benchmark itself, since all AIME years are math competitions. It falls out of niche-indexed competence $q_i(z)$ updating on task subtype tags.

Specialists emerge only when their niche arrives. math_hard specialists accumulate from the start, whereas the AIME'23 specialist activates at the AIME'22–'23 boundary and the AIME'24 specialist has zero anchor count until AIME'24 tasks arrive. Specialization is not pre-assigned but surfaces on demand as the competence landscape shifts.

Knowledge flows in structured channels, not uniformly. CODREAM insights concentrate on a few specific giver $\rightarrow$ recipient cells rather than spreading uniformly. The top givers are the same experts that dominate anchor assignments, and the heavy recipient columns belong to experts that are strong on a different niche, so every expert occupies both roles across the stream.

Leadership rotates and concentration rises with task difficulty. Leaders rotate over the Hard Math phase, and a different expert takes each AIME year. The specialization index rises from $\sim$0.1 on Hard Math toward $\sim$0.3 on AIME'24: the pool concentrates on a single anchor exactly where tasks are hardest.

Taken together, the gains in Tables 2–3 do not come from a fixed assignment of experts to niches, but from a continuously updating pool state that routes each task to agents whose competence fits the current niche.

5 Conclusion

We have argued that multi-agent test-time evolution is fundamentally different from single-agent evolution replicated N times. Beyond individual context and memory, a multi-agent system evolves who collaborates, how they collaborate, and how knowledge flows across the population. These team and population components have no single-agent counterpart and give rise to emergent phenomena that no individual learner can express. EVOCHAMBER instantiates all three evolution levels over a coevolving agent pool without gradient updates, with CODREAM as its core mechanism for verified asymmetric knowledge transfer. Across three heterogeneous task streams and two model families, EVOCHAMBER consistently outperforms all baselines. The most striking is what emerges without being engineered: N identical agents spontaneously differentiate into several stable niche specialists, leadership rotates across domains, and knowledge flows through structured channels rather than uniformly. This pattern is reproducible across random seeds even as the identity of each specialist changes, confirming emergent specialization as a structural consequence of multi-agent evolution. Limitations and future directions are discussed in Appendix A.

Acknowledgments and Disclosures

The authors received no third-party funding in direct support of this work and declare no competing interests.

References

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

[2] Yurui Chang, Yiran Wu, Qingyun Wu, and Lu Lin. Memcollab: Cross-agent memory collaboration via contrastive trajectory distillation, 2026. URL https://arxiv.org/abs/2603.23234.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

[4] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*, 2024.

[5] Yixing Chen, Yiding Wang, Siqi Zhu, Haofei Yu, Tao Feng, Muhan Zhang, Mostofa Patwary, and Jiaxuan You. Multi-agent evolve: LLM self-improve through co-evolution, 2025. URL https://arxiv.org/abs/2510.23595.

[6] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

[7] Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*, 2024.

[8] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378, 2019.

[9] Wenzhe Fan, Ning Yan, and Masood Mortazavi. EvoMem: Improving multi-agent planning with dual-evolving memory, 2025. URL https://arxiv.org/abs/2511.01912.

[10] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[11] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework, 2024. URL https://arxiv.org/abs/2308.00352.

[12] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025.

[13] Yue Hu, Yuzhu Cai, Yaxin Du, Xinyu Zhu, Xiangrui Liu, Zijie Yu, Yuchen Hou, Shuo Tang, and Siheng Chen. Self-evolving multi-agent collaboration networks for software development. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4R71pdPBZp.

[14] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[15] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[16] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

[17] Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024.

[18] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023.

[19] Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*, 2024.

[20] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[21] OpenAI. GPT-4 technical report, 2024. URL `https://arxiv.org/abs/2303.08774`.

[22] OpenAI. GPT-4.1 family, 2025. URL `https://openai.com/index/gpt-4-1/`.

[23] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL `https://arxiv.org/abs/2310.08560`.

[24] Chanwoo Park, Seungju Han, Xingzhi Guo, Asuman E Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. MAPoRL: Multi-agent post-co-training for collaborative large language models with reinforcement learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30215–30248, 2025.

[25] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 15174–15186, 2024.

[26] Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large language model-based multi-agent collaboration. In *The Thirteenth International Conference on Learning Representations*, 2025.

[27] Jon Saad-Falcon, Adrian Gamber, and Christopher Ré. Archon: An architecture search framework for inference-time techniques. In *The Thirteenth International Conference on Learning Representations*, 2025.

[28] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=vAElhFcKW6`.

[29] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *The Thirteenth International Conference on Learning Representations*, 2025.

[30] Qwen Team. Qwen3 technical report, 2025. URL `https://arxiv.org/abs/2505.09388`.

[31] Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities, 2024. URL https://arxiv.org/abs/2406.04692.

[32] Kun Wang, Guibin Zhang, ManKit Ye, Xinyu Deng, Dongxia Wang, Xiaobin Hu, Jinyang Guo, Yang Liu, and Yufei Guo. MAS^2: Self-generative, self-configuring, self-rectifying multi-agent systems. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=qumy27hMDY.

[33] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

[34] Yinjie Wang, Ling Yang, Guohao Li, Mengdi Wang, and Bryon Aragam. ScoreFlow: Mastering LLM agent workflows via score-based preference optimization, 2025. URL https://arxiv.org/abs/2502.04306.

[35] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

[36] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

[37] Xiangyuan Xue, Yifan Zhou, Guibin Zhang, Zaibin Zhang, Yijiang Li, Chen Zhang, Zhenfei Yin, Philip Torr, Wanli Ouyang, and LEI BAI. CoMAS: Co-evolving multi-agent systems via interaction rewards. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ihwAzktmWc.

[38] Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. Agentnet: Decentralized evolutionary coordination for LLM-based multi-agent systems. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=tXqLxHlb8Z.

[39] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380, 2018.

[40] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[41] Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. Evoagent: Towards automatic multi-agent generation via evolutionary algorithms. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6192–6217, 2025.

[42] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025.

[43] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642, 2024.

[44] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024.

Appendix

A Limitations and Future Work

Limitations. We validate on two model families. Evaluating additional architectures would strengthen generalizability, though we expect the mechanism to transfer since it operates entirely through prompts with no architecture-specific components. The inference cost is roughly $3.6\times$ that of a single agent, which may be prohibitive in latency-sensitive settings, although EVOCHAMBER is more accurate than SC with $k{=}5$ at 72% of SC's token budget. The lifecycle operators rely on fixed thresholds that transfer across all streams without tuning, but learning them through meta-optimization would be preferable.

Future work. Stronger backbones and longer streams beyond 1000 tasks would enable studies of scaling limits, long-horizon specialization stability, and insight obsolescence. Formalizing role-conditioned credit attribution beyond the current shared team reward is another direction enabled by the three-level decomposition.

B More Experiments

Table 6: Multi-seed specialization metrics on the Hard Math Stream. Across three independent runs with different random seeds, the mean specialization index, peak concentration, and pool expansion (unique anchors from $N{=}20$ initial) are reproducible, while the *identity* of the specialist for each niche changes with the seed. This separates path-dependent identity from seed-invariant pattern.

Run (seed)	Mean spec. index	Max spec. index	Unique anchors	math_hard top-1 (distinct?)
Default order	0.131	0.313	33	specialist α (26%)
Shuffle, seed 42	0.114	0.212	42	specialist β (14%)
Shuffle, seed 123	0.123	0.291	40	specialist γ (26%)
Mean $\pm$ spread	0.123 ± 0.008	0.272 ± 0.050	38 ± 5	all three distinct

Pattern is seed-invariant; identity is seed-dependent. Table 6 reports specialization metrics across three independent runs. The specialization index and pool expansion are reproducible across seeds: mean 0.123 ± 0.008, unique anchors 38 ± 5 from an initial $N{=}20$. However, the specific agent that becomes each niche's specialist is disjoint across the three seeds. The pattern that each niche produces a dominant specialist is a consequence of niche-conditioned selection acting on a shared pool, while the identity of that specialist reflects symmetry breaking at cold start. This separation of seed-invariant pattern from path-dependent identity is a structural signature of multi-agent evolution that single-agent learners cannot produce.

Table 7: Pool size sensitivity on the Hard Math Stream. Both runs use Qwen3-8B with thinking mode, team size $k{=}3$, seed 42, and identical hyperparameters except the initial pool size N.

Config	math_hard	AIME'22	AIME'23	AIME'24	AIME'25	Overall
$N{=}3$	0.740	0.433	0.333	0.433	0.333	0.628
$N{=}20$	0.763	0.400	0.333	0.433	0.300	0.639
Δ	-0.023	$+0.033$	0.000	0.000	$+0.033$	-0.011

Pool size has minimal impact on final accuracy. Table 7 compares $N{=}3$ and $N{=}20$ on the Hard Math Stream. The overall gap is only 0.011 absolute, concentrated on math_hard. On all four AIME years, $N{=}3$ matches or slightly exceeds $N{=}20$. The two configurations also converge in pool dynamics: $N{=}3$ grows from 3 to 8 active agents via genesis during the AIME phase, while $N{=}20$ retains only 9 routinely selected agents by the end of the stream, so the effective pool sizes are comparable at convergence. Genesis fires a similar number of times under both configurations, 5 for $N{=}3$ and 4 for $N{=}20$, confirming that lifecycle operators adapt to the current pool state rather than depending on the initial size. This robustness suggests that the evolution mechanism, not the starting roster, is what drives performance.

C Related Work Positioning Table

Table 8 provides a structured comparison of EVOCHAMBER against representative prior methods along five design axes: whether the method is training-free, whether it maintains a pool of agents, whether knowledge transfers across agents, whether that transfer is asymmetric, and whether evolution is continuous over a task stream. EVOCHAMBER is the only method that satisfies all five criteria simultaneously.

Table 8: Positioning of EVOCHAMBER against representative prior methods. ✓ = fully satisfied; × = not satisfied; ○ = partially satisfied; — = not applicable.

Method	Training-free	Pool-level	Cross-agent	Asymmetric	Continuous
DyLAN, AutoGen, MetaGPT	✓	✓	—	—	×
AFlow, ScoreFlow	✓†	×	—	—	×
Reflexion, MemGPT [23], EvoAgent	✓	×	×	—	✓
AgentNet	✓	✓	×	—	✓
EvoMem (pool Reflexion)	✓	✓	×	—	✓
MemCollab	✓	✓	✓	×	✓
MAS^2	×	✓	✓	○	✓
EVOCHAMBER (full)	✓	✓	✓	✓	✓

†AFlow's MCTS requires hundreds of offline LLM calls per domain; the resulting workflow is frozen at inference time.

D Experience Archive Design Justification

As described in Section 3.3, each agent maintains two stores that separate reasoning insights by scope.

Subtask-level lessons are indexed by niche label and capture domain-scoped strategies, such as a proof technique for combinatorics or a debugging pattern for recursive algorithms. These lessons are retrieved by cosine similarity over task embeddings when the agent encounters a task in the same niche, providing targeted in-context guidance. Near-duplicate entries are merged via LLM-based deduplication to control redundancy.

Cross-domain meta-insights form a single pool not tied to any niche, capturing higher-order self-corrections such as "decompose the problem into sub-steps independently." Without a dedicated cross-domain store, an agent that learns careful decomposition from math cannot transfer this principle to code or QA without re-discovering it.

Both stores grow with the agent's full history, with no fixed capacity limit. At solve time, entries from both stores are retrieved by cosine similarity and prepended to the prompt as in-context guidance.

Separating niche-specific from cross-domain insights serves two purposes. First, it prevents tactical noise from polluting general metacognition. Second, it gives CODREAM the granularity needed to route each insight to the right audience: niche-local strategies are sent only to deficit agents on that niche, while cross-domain insights can propagate more broadly.

E Implementation Details

E.1 Operational Details

We provide concrete definitions for the quantities referenced in §3.

Style overlap ω_{ij}. Each agent maintains a dictionary mapping subdomain tags to its running competence on that subdomain. The style overlap ω_{ij} is the cosine similarity of these two competence vectors, aligned over the union of both agents' subdomain keys. When a subdomain tag appears in one agent's dictionary but not the other, the missing entry is treated as zero competence. High overlap indicates that two agents have developed similar skill profiles across the same set of subdomains, meaning they would contribute redundant perspectives to a team. Low overlap can arise either because the agents specialize in different subdomains or because one agent has been exposed to subdomains that the other has not encountered.

Pair synergy $\sigma_{ij}(z)$. The pair synergy on niche z is the mean team reward on past niche-z tasks in which agents i and j both participated. It is initialized to 0 and remains at 0 until the pair has co-participated in at least five niche-z tasks, avoiding noisy estimates from small samples. Synergy captures whether two agents complement each other on a specific niche: a pair that consistently achieves higher team rewards than either agent's solo competence would predict has high synergy.

Lifecycle operators. All four operators are evaluated every $\tau = 10$ tasks.

Fork targets agents in the top 10% by rolling-average reward. The operator clones the selected agent and mutates the clone's persona via a one-shot LLM call that instructs the backbone to emphasize the parent's dominant subdomain while preserving the parent's general role description. The clone inherits a copy of the parent's full memory store but receives a distinct agent ID, so subsequent competence updates diverge. Fork serves as controlled exploration in persona space: it amplifies successful strategies while introducing variation that may discover adjacent niches.

Merge fires when a pair's profile cosine similarity exceeds 0.95 and both agents have accumulated at least 10 tasks. The two agents are consolidated into a single agent that inherits both memory stores, with near-duplicate entries deduplicated via the same LLM-based deduplication used during normal insight injection.

Prune retires agents that have scored below $0.8\times$ the pool mean for 10 or more consecutive tasks. Pruned agents are removed from the pool entirely and their memory stores are discarded.

Genesis is triggered when the pool size drops below 15 agents or when no existing agent has niche affinity greater than 0.4 on a newly encountered task type. New agents are seeded with domain-specific personas generated by an LLM call that describes the uncovered niche, but with empty memory stores, so all subsequent knowledge must be earned through task experience.

E.2 Inference Configuration

Serving. All experiments use Qwen3-8B served locally via vLLM [14] with two instances under round-robin load balancing. Key parameters: tensor parallel size 1 GPU per instance, max model length 32,768 tokens, GPU memory utilization 0.90, max batch size 32, dtype bfloat16.

Generation mode. All three streams use thinking mode.

Token budgets. Task solving uses 4,096 output tokens per agent, increased to 8,192 for Hard Math with thinking. Each CoDream phase uses 2,048 tokens per agent. Profile injection prepends retrieved insights to the system prompt.

E.3 Hyperparameters

Table 9 lists all hyperparameters. A single configuration is used across all three streams and both model families with no per-benchmark tuning.

Table 9: EvoChamber hyperparameters. A single configuration is used across all streams and backbones.

Hyperparameter	Value	Notes
Pool size N	20	Fixed across all streams
Team size k	3	Greedy one-at-a-time selection
EWMA decay α	0.3	For $q_i(z)$ competence update
$q_i(z)$ initialization	0.5	Prior to first niche encounter
Complement weights $(\lambda_q, \lambda_\sigma, \lambda_\omega)$	$(1.0, 0.3, 0.5)$	Competence / synergy / style-overlap penalty
Scout weights (λ_u, λ_d)	$(0.3, 0.5)$	Under-exposure / diversity penalty
Lifecycle interval τ	10 tasks	Fork / merge / prune / genesis check
Fork threshold	Top 10% by rolling average	Specializes high performers
Merge threshold	Profile cosine sim > 0.95, $\geq$10 tasks each	Collapses redundant agents
Prune threshold	Below $0.8\times$ pool mean for $\geq$10 consecutive tasks	Retires persistent underperformers
Genesis trigger	Max niche affinity $<$0.4	Seeds domain-specific new agents
CoDream trigger θ	0.6	Team reward threshold
Insight dedup cosine	0.85	Prevents near-duplicate insights
Deficit gate	Below-median recent performance	For asymmetric routing

E.4 Evaluation Protocol

All streams use a fixed task order across methods to ensure comparable learning trajectories.

All agents are initialized with a generic helpful-assistant persona; domain-specific knowledge emerges entirely from task experience.

F CoDream Isolation Experiment

The main-text ablation in Table 5 removes CoDream from the full system while keeping all other components. Here we run a more controlled isolation on a smaller scale to sharpen the conclusion. We select a 30-task math subsequence from AFlow-Stream and compare three configurations that differ in exactly one dimension: SA uses a single agent with no pool, EvoChamber w/o CoDream maintains the full 20-agent pool with individual experience accumulation, team composition, and lifecycle operators but disables cross-agent knowledge transfer, and EvoChamber (full) enables CoDream on top of the same pool infrastructure. The goal is to test whether pool infrastructure alone already improves over a single agent, or whether the improvement requires cross-agent transfer.

Table 10: Controlled CoDream isolation on a 30-task AFlow math subsequence.

Configuration	Accuracy
SA (no pool)	0.633
EvoChamber w/o CoDream	0.633
EvoChamber (full)	**0.700**

The key finding is that EvoChamber without CoDream matches SA exactly at 0.633. Maintaining 20 agents with individual experience, team composition, and lifecycle management produces zero gain over a single agent when cross-agent knowledge sharing is absent. This result is expected on a short, single-domain subsequence: without CoDream, each agent accumulates experience independently, and the team composition operator can select competent agents but cannot transfer knowledge from strong agents to weak ones. The pool infrastructure provides the scaffolding for knowledge flow, but it is CoDream that activates the flow.

Adding CoDream yields 0.700, a $+10.5\%$ relative improvement, confirming that asymmetric transfer is the mechanism responsible for the multi-agent advantage on this subset. On the full 600-task AFlow-Stream, the gap is even larger: removing CoDream causes a -0.108 drop in overall accuracy (Table 5), with the effect concentrated on dependent-reasoning domains such as HotpotQA and DROP where cross-agent coordination knowledge is most valuable.

G Hard Code Stream Per-Benchmark Breakdown

Table 3 in the main text omits the HumanEval column because HumanEval saturates at 1.000 for every method in our harness. Overall is the micro-average over all 586 tasks including HumanEval. Table 11 provides the full per-benchmark breakdown.

Table 11: Hard Code Stream per-benchmark accuracy. HumanEval saturates at ceiling for all methods. Overall is the micro-average over all 586 tasks, matching Table 3.

Method	MBPP+	HumanEval	CodeContests	Overall
SA	0.842	1.000	0.068	0.667
SC (k=5)	0.849	1.000	0.198	0.708
DyLAN	0.825	1.000	0.189	0.695
AgentNet	**0.887**	1.000	0.102	0.698
EvoMem	0.885	1.000	0.027	0.672
MemCollab	0.870	1.000	0.084	0.682
EvoChamber (full)	0.861	1.000	**0.352**	**0.757**

MBPP+ clusters near 0.85 for all multi-agent methods, leaving CodeContests as the discriminating subset. We inspected HumanEval task-level outputs and confirmed that all methods solve every problem correctly; the remaining minor diversity across runs is within grading tolerance.

On CodeContests, EVOCHAMBER achieves 0.352, a $1.8\times$ improvement over SC k=5 and $3.5\times$ over AgentNet. The mechanism is experience-guided debugging: agents whose experience archives contain prior failure patterns and repair strategies for similar problem classes attempt more targeted corrections on subsequent CodeContests problems. This is a direct consequence of cross-difficulty transfer within the stream, as debugging patterns first learned on easier MBPP+ problems accumulate in agent profiles and propagate to deficit agents via CODREAM before the harder CodeContests problems arrive.

H Order and Execution Robustness: Setup

This section provides setup details for the robustness experiments reported in Table 5. All runs use the same Qwen3-8B backbone, the same pool and team sizes, and the same code version as the main Hard Math Stream result in Table 2.

Shuffle conditions. The default task order presents domains in sequential blocks as described in §4.1. The two shuffle conditions reorder all 382 tasks across domains using the given random seed, producing a different task ordering while preserving the same task set.

Uniform execution. This condition disables LEADLEARN's dynamic structure selection and forces voting for every task, using the same self-consistency implementation as the SC baseline for each team member. All other components remain intact: individual experience accumulation, CODREAM, and lifecycle operators.

SA reference. The SA score in the table is the default fixed-order result. Since SA does not accumulate experience across tasks, its performance under any task permutation is statistically indistinguishable from the fixed-order score. We verified this on a single shuffled run with seed 42, which produced equivalent results.

I Per-Subset Regime Analysis

EVOCHAMBER's per-subset gains vary by the per-agent success rate on that subset, with a regime structure that matches the underlying mechanism.

Very high accuracy ($\geq$80%). When the backbone already solves most tasks, such as GSM8K, MBPP, or MATH Level 3 with strong backbones like GPT-4.1-mini on math_hard at 0.824, there is little room for improvement and gains are modest, ranging from $+0.01$ to $+0.07$. In this regime, the dominant contribution comes from team diversity and the leader's dynamic structure selection rather than from cross-agent distillation.

Mid accuracy (40%–70%). MATH Level 4/5 with Qwen3-8B, where math_hard base accuracy is 0.302, is the design sweet spot for CODREAM: enough verified solutions for reliable crystallization, yet wide gaps between struggling and successful agents. Gains here range from $+0.07$ on Qwen3-8B math_hard to $+0.20$+ on GPT-4.1-mini AIME'22 where base accuracy is 0.40, and the cross-agent distillation component contributes a meaningful share.

Low accuracy (20%–40%). AIME-level tasks under Qwen3-8B yield per-agent success rates near 15–20%, and under GPT-4.1-mini near 30–40%. Verified solutions are rare but not absent. For Qwen3-8B, CODREAM's direct contribution on AIME is small and statistically noisy at this backbone capability. The full EVOCHAMBER system still lifts AIME via team selection and lifecycle. For GPT-4.1-mini, the same AIME subsets land in a higher base-accuracy regime of 30–40%, and we observe the largest per-subset gains of the entire paper, $+0.20$ to $+0.27$. This is consistent with the mechanism: with more frequent verified solutions, cross-agent distillation has more material to crystallize and route.

Very low accuracy (<15%). No effect is expected: at 10% per-agent accuracy with team of 3, the probability that at least one agent succeeds is $1 - (0.90)^3 = 0.271$, and the probability of a verified solution on two independent attempts drops quickly. In practice we observe CODREAM correctly abstains when no agent solves a task.

Self-Consistency collapse derivation. With five independent agents at 20% accuracy, the probability of majority-correct:

$$P(\text{majority correct}) = \sum_{j=3}^{5} \binom{5}{j} (0.20)^j (0.80)^{5-j} \approx 0.058 = 5.8\%.$$

This is *lower* than 20% single-agent: majority voting actively overrides rare correct answers. The empirical SC result of 0.067 on AIME matches this prediction. EVOCHAMBER avoids this failure mode because its team leader selects non-voting structures such as debate, generator-critic, or decompose when the base rate is low and rare successes exist.

J Case Study: How EVOCHAMBER Learns Competition Mathematics

We trace specific events from the Hard Math Stream rerun used in Table 2, conducted with Qwen3-8B on seed 42 over 382 tasks. All events below are parsed directly from the per-task run log. Agent IDs are real, truncated to 8 hex characters.

Early expert identification, tasks 10–30 on math_hard. The first lifecycle events fire at task 10, after the rolling performance window has stabilized. All three events are simultaneous *specialize*/fork events on agents `6f3dcc14`, `bb411e98`, and `119b9e09`, each with sustained mean reward of 0.80–1.00 on math_competition_hard. These same three agents are forked again at tasks 20, 30, 50, and most other lifecycle checkpoints in the math_hard phase. By the end of the math_hard phase at task 261, these three account for all 43 specialize events in the entire stream.

Insight crystallization on math_hard. The first CODREAM trigger occurs at task 11 with team score 0/3. Three insights are crystallized simultaneously by the same three agents `6f3dcc14`, `bb411e98`, `119b9e09`. A representative insight:

> *"When counting integers in a range divisible by multiple numbers, use the inclusion-exclusion principle with LCM adjustments: (1) compute LCM of all divisors; (2) use inclusion-exclusion to count numbers divisible by subsets of divisors; (3) alternate signs."*

Over the math_hard phase, these same three agents produce 72 of 93 verified insights in the run, or 77% of the total. The insights are concrete competition-math techniques: generating functions for recursive growth, Möbius inversion for overlapping-set counts, Chinese Remainder Theorem for systems of congruences.

Quality degradation on AIME. CODREAM fires 7 times in the AIME phase, 4 on AIME'22 and 3 on AIME'23, with zero triggers on AIME'24 or AIME'25. The character of the insights changes. At task 267, `aime_2022_5` with team score 0, the three insights crystallized are not math strategies but meta-advice about extracting numerical values from text:

> *"When a problem involves extracting and reconciling numerical data from a passage with multiple steps or implicit relationships, create a structured checklist of required values . . . "*

When no team member solves the underlying math problem, the crystallize step has no successful trajectory to distill from, and the agents' reflections produce generic reading-comprehension advice rather than targeted mathematical techniques. This is consistent with CODREAM's regime condition in §I: on very hard problems where per-agent accuracy is too low for any team member to succeed, the mechanism cannot extract useful material. The verification gate still passes these candidates because the re-attempt with meta-advice applied happens to score marginally higher than the original failure, but their contribution to future AIME performance is marginal.

Late-stream lifecycle: from forking to genesis. Specialize events stop entirely at the math_hard/AIME boundary, task 262. In the AIME phase, the system instead fires 5 *genesis* events seeding new agents in response to "coverage gap for task type `aime_problem`, max affinity = 0.20" and 1 *prune* removing an agent with 6 consecutive underperforming tasks. **Summary: emergent structure from identical initialization.** All 20 agents start from the same backbone with empty insight stores. Over 382 tasks, CODREAM fires 34 times and crystallizes 93 verified

insights. These insights are not evenly distributed: seven agents contribute all 93, and the top three, `bb411e98`, `119b9e09`, and `6f3dcc14`, contribute 72 of 93, or 77% of the total. Independently, lifecycle specialization events concentrate on the same three agents: these three account for all 43 specialize events, with `bb411e98` at 16, `6f3dcc14` at 14, and `119b9e09` at 13. The top insight contributors and the top forked agents overlap completely. An expert core differentiates from the pool purely through environment feedback.

The system's lifecycle behavior also splits cleanly by regime: all 43 specialize events occur during the math_hard phase where per-agent base accuracy is sufficient to identify clear top performers and fork them for controlled exploration. During the AIME phase, specialize events cease and lifecycle shifts to a different mix: 5 genesis events for coverage gaps and 1 prune for consecutive underperformance. This mirrors the regime analysis in §I: at mid base accuracy the system can identify and amplify specialists, while at very low base accuracy it instead seeds new agents and retires unproductive ones.

K Lifecycle Operator Analysis

Table 12: Lifecycle contribution by stream phase on AFlow-Stream.

Configuration	Early (1–200)	Mid (201–400)	Late (401–600)
EvoChamber (full)	0.868	0.876	0.879
−Lifecycle	0.867	0.869	0.871
Δ	+0.001	+0.007	+0.008

Lifecycle contribution is negligible early and grows modestly in mid-to-late phases. Fork/merge/retire maintain diversity and prune stagnation over long streams, not accelerate early learning.

L CoDream Insight Examples

Representative insights crystallized during actual experimental runs, lightly edited for brevity. Each insight was generated by a specific agent during a post-task reflection session, verified by re-attempt, and routed to the appropriate experience archive.

L.1 Math Insights, Hard Math Stream, Qwen3-8B

Example M1: modular arithmetic, from task math_hard_10.

> *"Modular arithmetic constraints must be integrated into the sequence's structural definition rather than treated as external constraints. This integration allows for a more accurate modeling of sequences where the modulus influences the sequence's recursive or periodic behavior."*

Example M2: trapezoid geometry, from task math_hard_13.

> *"The correct application of the trapezoid area formula hinges not just on identifying parallel sides, but also on accurately measuring the perpendicular height. A structured geometric analysis, starting with side identification, followed by precise height measurement, and finally applying the formula, prevents formula misapplication."*

Example M3: constraint-graph reformulation, from AIME 2022.

> *"For constraint-satisfaction problems, model as a graph where nodes represent constraints and edges represent interactions; this allows more efficient traversal and resolution of complex dependencies than direct decomposition."*

L.2 Code Insights, Hard Code Stream, Qwen3-8B

Example C1: memoization with state compression, from MBPP task 54.

"When a combinatorial problem has high symmetry or complex dependencies, use memoization with state compression: represent the state as a tuple of essential parameters, and cache results to avoid redundant computation."

Example C2: bounded arithmetic with saturation, from CodeContests task 425.

"When input validation involves numerical ranges and potential overflow, use bounded arithmetic with explicit saturation: clamp intermediate values to the valid range [min_val, max_val] using `min(max(value, min_val), max_val)` *before further computation."*

Example C3: symbolic + numerical cross-validation, from MBPP task 17.

"When symbolic computation verifies mathematical logic with potential edge cases, cross-validate with numerical evaluation at specific test points: (1) define test points covering edge cases and typical scenarios; (2) evaluate the symbolic expression numerically; (3) compare symbolic and numerical results to catch simplification errors."

These examples illustrate the style of crystallized insights: actionable, cross-task patterns rather than problem-specific hints. In the current implementation, most insights are classified as cross-domain by the crystallization step. Niche-specific routing is exercised when the insight-classification prompt assigns lower transferability, and the asymmetric sharing decision between selective and broadcast routing is additionally gated by the cosine-similarity deficit check at injection time.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Abstract and introduction state three specific claims: (1) three-level co-evolution framework, (2) accuracy gains on three streams with ablation attributing the largest drop to CoDream, (3) emergent specialization from identical initialization. All are supported by experiments in Section 4. Limitations are discussed in Appendix A.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The conclusion points to Appendix A, which explicitly discusses limitations: only two model families validated, $3.6\times$ inference cost, and fixed lifecycle thresholds. Future work further identifies shared reward attribution as an open problem. Appendix J analyzes the verification gate's weakness on hard tasks.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not include theoretical results or proofs. All claims are supported empirically.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 4.1 specifies datasets, baselines, metrics, and backbone configuration. Section 3.1 describes the full solve-evolve loop. Appendix E provides all hyperparameters (Table 9), inference configuration (Section E.2), operational definitions (Section E.1), and evaluation protocol (Section E.4). A single hyperparameter configuration is used across all streams with no per-benchmark tuning.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The public implementation at `https://github.com/Mercury7353/EvoChamber` includes installation instructions, dependencies, benchmark loaders, and commands for the three paper streams and baseline conditions. The manuscript specifies the evaluation setup in Section 4.1 and hyperparameters in Appendix E.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.1 specifies datasets, task counts, niche definitions, baselines, backbone model, and metrics. Appendix E provides all hyperparameters (Table 9), inference configuration (Section E.2), serving details, token budgets, and evaluation protocol (Section E.4). A single configuration is used across all streams with no per-benchmark tuning.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Main accuracy tables report single-run results due to the computational cost of full-stream evaluation (382–600 tasks per stream with 20 agents). Table 6 reports multi-seed specialization metrics across three seeds showing reproducible emergent patterns, and Section 4.3 tests robustness to shuffled task orderings, but per-method error bars are not provided.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 4.1 identifies the H100 GPU and API-based backbone, and Appendix E.2 reports serving and token-budget settings.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses publicly available benchmarks, involves no human subjects, and poses no foreseeable ethical concerns.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work is foundational research on multi-agent coordination for math and code benchmarks. It does not introduce new models or capabilities with direct negative societal applications beyond those inherent to the underlying LLMs.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release models or datasets with high misuse risk. It proposes a coordination framework on top of existing LLMs.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The benchmarks and model providers are credited in Section 4.1, and vLLM is cited in Appendix E.2.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The new framework implementation is publicly documented at `https://github.com/Mercury7353/EvoChamber`; its README describes installation, the three benchmark streams, experiment commands, hyperparameters, and the repository structure. The code is released under the MIT license; no new dataset or pretrained model weights are introduced.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: No human subjects or crowdsourcing involved.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: No human subjects research involved.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are a core component of the method: agents solve tasks, perform collaborative reflection and insight distillation, and generate lifecycle persona variants. Section 3, Section 4.1, and Appendix E describe these uses, the Qwen3-8B and GPT-4.1-mini backbones, and inference settings.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.